DOCUMENT RESUME

ED 097 024 IR 001 211

AUTHOR	Yu, Clement T.
TITLE	A Formal Construction of Term Classes. Technical Report No. TR73-18.
INSTITUTION	Alberta Univ., Edmonton. Dept. of Computing Science.
REPORT NO	AU-DCS-TR-3-18
PUB DATE	Dec 73
NOTE	56p.
EDRS PRICE	MF-$0.75 HC-$3.15 PLUS POSTAGE
DESCRIPTORS	Computer Programs; *Computers; *Computer Science; Data Bases; Information Processing; *Information Retrieval; *Information Science; Information Storage; Information Systems; Mathematical Models
IDENTIFIERS	Term Classes

ABSTRACT

The computational complexity of a formal process for the construction of term classes for information retrieval is examined. While the process is proven to be difficult computationally, heuristic methods are applied. Experimental results are obtained to illustrate the maximum possible improvement in system performance of retrieval using the formal construction over simple term retrieval. (Author)

ED 097021

BEST COPY AVAILABLE

A FORMAL CONSTRUCTION OF TERM CLASSES.

By

Clement T. Yu

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Technical Report TR73-18

December 1973

IR 001 211

DEPARTMENT OF COMPUTING SCIENCE

The University of Alberta

Edmonton, Alberta, Canada

Summary*

The computational complexity of a formal process for the construction of term classes is examined. While the process is proved to be difficult computationally, heuristic methods are applied. Experimental results are obtained to illustrate the maximum possible improvement in system performance of retrieval using the formal construction over simple term retrieval.

*This research was supported in part by National Science Foundation Grants GN27224 and GJ33171X.

1. Introduction

Many experiments have been performed to examine the application of term classifications to information retrieval. It has been shown by Salton [1], Spark, Jones and Jackson [2] and many others that a substantial improvement in system performance is obtained over simple term retrieval by using automatic thesaurus construction. The construction is generally based on the co-occurrence of terms in documents. It has been pointed out by Doyle [3] that such construction may have a number of difficulties. To avoid these difficulties, Jackon [4] has proposed a formal construction of term classes known as pseudo-classification in which the correlations of relevant documents which are not retrieved are raised and those of the irrelevant documents which are retrieved are lowered. Such construction is based on a given correlation function, a set of requests and the relevance judgement of the documents with respect to these requests and is outlined as follows. If a document is relevant to a request but the correlation between the document and the request is not sufficiently high for the document to be retrieved, then it is likely that some of the terms used in the request, though similar in meaning to, are distinct from those in the document. If term classes are formed such that two distinct terms in the same class constitute a class 'match', then the terms in the request and those in the document which are related semantically may score

enough class 'matches' so as to raise the correlation between the document and the request and thus enable the document to be retrieved. Similarly, if a document is irrelevant to a request but the correlation between the document and the request is high enough for the document to be retrieved, then terms in the query and those in the document are likely to score some class 'mismatches' to bring down the correlation so as to preclude the retrieval of the document. It is hoped that the construction will bring about the situation where every relevant document is retrieved and every irrelevant document is rejected. As pointed out by Jackson [4], whether a classification derived from a given set of queries and a given set of relevance judgement agrees with that derived from another set of queries and another set of relevance judgement has yet to be explored.

In this paper, the computational complexity of the formal construction is examined. While the construction is shown to be 'difficult' computationally, heuristic methods are tried on a set of data to find out what improvement this classification model can possibly have over simple term retrieval.

2. Problem Definition

We shall assume all document and request vectors are binary. The formal construction of term classes will be formulated as follows.

Definition 2.1. A document D is retrievable (not retrievable) by a request R if $f(D,R) > T$ ($f(D,R) \leq T$), where T is a pre-set constant, and f is a matching function measuring the closeness between the document D and the request R. Normally, the value of f is determined by the number of terms in common and the number of terms not in common between the D & the R.

However, if we want f to also measure the similarity in meaning of the terms in D and in R, f will have to satisfy some more properties. The exact properties which f should possess will be given later.

Definition 2.2. If $D = (a_1,a_2,\ldots,a_m)$ and $R = (b_1,b_2,\ldots,b_m)$ are two binary vectors, then

$$D \cap R = (c_1,c_2,\ldots,c_m) ,$$
$$D - R = (d_1,d_2,\ldots,d_m)$$
$$R - D = (e_1,e_2,\ldots,e_m) \quad \text{and}$$
$$D \cup R = (f_1,f_2,\ldots,f_m)$$

are defined as follows.

$$c_i = \min\{a_i,b_i\}$$
$$d_i = a_i \dot{-} b_i \quad \dagger$$
$$e_i = b_i \dot{-} a_i \quad \dagger$$
$$f_i = \max\{a_i,b_i\}$$

$\dagger a \dot{-} b = \begin{cases} a - b & \text{if } a \geq b \\ 0 & \text{otherwise.} \end{cases}$

We now introduce term "classes" with the intention of putting terms in D-R and those in R-D which are similar in meaning into the same term "classes". We assume certain subsets of the set of index terms to be called "classes". These may overlap, so that a term can belong to more than one class. With each term t we associate a vector

$$t = (t_1, t_2, \ldots, t_k) \text{ where}$$

$$t_i = \begin{cases} 1 \text{ if term i is in the } i^{th} \text{ class} \\ 0 \text{ if term i is not in the } i^{th} \text{ class} \\ d \text{ if don't care or don't know.} \end{cases}$$

The word "class" will remain undefined and we shall assume that it is synonymous to the word "interpretation". Since a term may have more than one interpretation, it may have more than one "1" in its k-ternary vector. This is consistent with our ordinary usage of words.

Let us call this vector, the class vector of the index term t.

Definition 2.3. Given two index terms A and B and their k-ternary class vectors $(a_1, a_2, \ldots, a_k)$ and $(b_1, b_2, \ldots, b_k)$, we define the class vector representing $A \cup B$ by $C = (c_1, c_2, \ldots, c_k)$ where

$$c_i = \begin{cases} 1 \text{ if either } a_i = 1 \text{ or } b_i = 1 \\ 0 \text{ if neither } a_i = 1 \text{ nor } b_i = 1 \\ \quad \text{and at least one of them} = 0 \\ d \text{ if } a_i = b_i = d. \end{cases}$$

Note that the above definition makes a '1' dominate a '0'. This is natural because a positive response is much more important than a negative one.

Definition 2.4. The number of term matches between a document D and a request R is the total number of 1's in $D \cap R$.

The number of term mismatches between a document D and a request R is the total number of 1's in $(D-R) \cup (R-D)$.

Definition 2.5. The class match vector, $C = (c_1, c_2, \ldots, c_k)$, of two class vectors $A = (a_1, a_2, \ldots, a_k)$ and $B = (b_1, b_2, \ldots, b_k)$ is defined by

$$c_i = \begin{cases} 1 \text{ if } a_i = b_i \text{ and } a_i \neq d \\ 0 \text{ otherwise.} \end{cases}$$

Remark: If $a_i = b_i = 0$ i.e., a (0,0) condition, then there is a class match between A and B. However, if A and B are document vectors instead of class vectors, then there is no term matches according to definition 2.4. This is because in general there are too many (0,0) conditions in documents, while the number of class matches due to (0,0) conditions is a lot less. (See definition 2.3 and diagram 2.1 for the construction of class vectors for D-R and R-D). If we take the approach that a (0,0) condition is a match but is less important than a (1,1) condition for both term and class matches, then a slight modification of the theory developed later will carry over.

The class mismatch vector $D = (d_1, d_2, \ldots, d_k)$ of the two vectors is defined by

$$d_i = \begin{cases} 1 \text{ if } a_i \neq b_i \text{ and neither } a_i = d \\ \quad \text{nor } b_i = d \\ 0 \text{ otherwise.} \end{cases}$$

The situation in which ($a_i = 0$, $b_i = d$) or ($a_i = d$, $b_i = 0$) or ($a_i = d$, $b_i = 1$) or ($a_i = 1$, $b_i = d$) or ($a_i = d$, $b_i = d$) corresponds to a no match condition, i.e., a no match is neither a class match nor a mismatch.

Definition 2.6. The number of class matches between two class vectors is given by the number of 1's in their class match vector. Similarly the number of class mismatches is given by the number of 1's in the class mismatch vector.

Based on the above definitions, we require f, the matching function, to be non-increasing in the number of class and term mismatches and non-decreasing in the number of class and term matches.

Remark: The reason why we impose the condition of non-decreasing instead of increasing in the number of class and term matches is that we want to include most natural matching functions in the class we defined. An example showing that the stricter condition is not appropriate is:

Let the function $f = \frac{a-b}{a+b} + \frac{a'-b'}{a'+b'}$ where a and b are the number of term matches and mismatches, respectively, and a' and b' are the number of class matches and mismatches. It is easily seen that when b' = 0, increasing

a' will not increase the value of f.

Similar remarks apply to the condition of non-increasing in the number of term and class mismatches.

When we calculate the matching function value f(D,R), we first have to find the total number of term matches and mismatches. We have to delete the terms in common between the D and the R when calculating the number of class matches and mismatches because any term in D∩R would automatically contribute considerably to the number of class matches and the effect would dominate that due to other terms.

A diagram illustrating how f(D,R) is calculated is shown in Diagram 2.1.

Definition 2.7. An assessment matrix, Z, of m documents and n requests is a binary matrix of order m×n. The assessment matrix Z will correspond to the user's relevance judgement, i.e.,

$$Z(i,j)=\begin{cases} 1 \text{ if the user judges that } D_i \text{ is relevant to } R_j \\ 0 \text{ otherwise.} \end{cases}$$

A (D,R) pair, say (D_i,R_j), satisfies assessment iff one of the following conditions is satisfied:

i) When $Z(i,j) = 1$, $f(D_i,R_j) > T$

ii) When $Z(i,j) = 0$, $f(D_i,R_j) \leq T$.

Step 1 Obtain $D \cap R$, $D - R$ and $R - D$.

Step 2 From $D \cap R$, obtain the number of term matches.

Step 3 From $D - R$ and the term classification matrix, obtain the class vector of $D - R$ (using Definition 3.1.3 iteratively), denoted by $(D - R)^C$.

Step 4 From $R - D$ and the term classification matrix, obtain the class vector of $R - D$, denoted by $(R - D)^C$.

Step 5 From $(D - R)$ and $(R - D)$, obtain the number of term mismatches.

Step 6 From $(D - R)^C$ and $(R - D)^C$ obtain the number of class matches and mismatches.

Step 7 From the number of class matches and mismatches and the number of term matches and mismatches, calculate f.

Diagram 2.1. To Calculate f(D,R).

Thus, if a (D,R) pair satisfies assessment, then D is retrievable by R iff D is relevant to R.

The problem we are trying to solve is posed as follows.

Given m documents, n requests, an assessment matrix Z, and a threshold T, find a k-tuple class vector for each index term such that as many (D,R) pairs satisfy assessment as given by the matrix Z as possible. This is the same as saying that as many of the relevant documents are retrieved and as many as the irrelevant documents are rejected as possible.

Let us call this problem the satisfiable assessment problem.

If there is a matrix C whose rows represent the index terms and whose columns represent the classes, then the above problem is the same as manipulating the entries of C so that as many entries of Z are satisfied as possible. Let the matrix C be called the term classification matrix. Usually, some of the terms are known to have certain semantic relations with some other terms, i.e., some of the entries of the matrix C are fixed. Thus, the satisfiable assessment problems reduces to fixing some entries of the matrix C, while varying other entries so that as many of the entries of Z are satisfied. Let us call this problem the satisfiable assessment problem with a partial solution. We shall show in the next section that the satisfiable assessment problem with a partial solution is 'difficult' computationally.

3. The Computational Complexity of the Problem.

[The results of this section were obtained jointly with S.K. Sahni.]

Cook [5] introduced an equivalent class of 'difficult' problems known as the polynomial complete problems. So far, no one has obtained a polynomial algorithm (i.e., an algorithm with the number of operations bounded by a polynomial function of the length of the input) for any one of these problems. However, all of these problems can be solved by a non-deterministic Turing machine in polynomial time. Whether there is deterministic polynomial algorithm for any of these problems depends on whether we can simulate an arbitrary non-deterministic Turing machine in polynomial time by a deterministic Turing machine running also in polynomial time. While we are unable to settle the above question, we will show that the satisfiable assessment problem with a partial solution is polynomial complete.

All of the above mentioned 'difficult' problems are polynomial reducible in the sense that if P_1 and P_2 are any two of these problems and P_1 can be solved in time $f(n)$, with input n, then P_2 can be solved in time $f(p(n))$ where p is a polynomial function. A polynomial complete problem is deciding whether a given formula in conjunctive normal form having at most 3 literals per clause is satisfiable (to be explained later). (see [5]). From this, one can easily show that satisfiability with exactly 3 literals per clause is polynomial complete, see [6]. Thus

all we need to claim that our problem is polynomial-complete is to show that there is a deterministic polynomial algorithm for our problem iff there is one for the exactly 3-literal satisfiability problem.

An example showing a formula in conjunctive normal form is as follows.

Example 3.1. $(X_1 \vee \bar{X}_2 \vee X_4) \wedge (X_5 \vee X_2 \vee \bar{X}_9 \vee X_{11}) \wedge (\bar{X}_1 \vee X_3 \vee X_9)$. This formula has three clauses, namely $(X_1 \vee \bar{X}_2 \vee X_4)$, $(X_5 \vee X_2 \vee \bar{X}_9 \vee X_{11})$ and $(\bar{X}_1 \vee X_3 \vee X_9)$. Each of the letters or its complement is called a literal. Thus X_1, $\bar{X}_2$ or X_{11} is a literal.

Definition 3.1. A formula is satisfiable iff under some assignment of 0-1 values to the variable, every clause of the formula has the value '1'.

In the above example, the assignment $x_1 = x_2 = x_3 = 1$ results in each clause having the value 1 and so the formula is satisfiable.

The main result of this section is:

Theorem 3.1. The satisfiable assessment problem with a partial solution is polynomial complete.

Proof: For a proof of this theorem see Appendix 1.

Since any problem which can be solved by a random access machine in deterministic polynomial time can also be simulated by a Turing machine in deterministic polynomial time, the above problem is also polynomial complete when considering a random access machine.

4. Heuristic Methods

In the last section, we showed that it was very likely that our problem would require exponential time. In this section, we shall present some heuristic methods which run in relatively little time as compared to an exhaustive enumeration algorithm but do not guarantee that the number of (D,R) pairs satisfying assessment would be the maximum possible.

Notation 4.1. Let $\{C_{ij}\}_{\substack{i=1,m \\ j=1,n}}$ be the term classification matrix where i denotes the term, j denotes the class. $Op(C_{ij},x\rightarrow y)$ means that C_{ij} is changed from x to y. There are six such operations, namely $Op(C_{ij},0\rightarrow 1)$, $Op(C_{ij},0\rightarrow d)$, $Op(C_{ij},1\rightarrow d)$, $Op(C_{ij},1\rightarrow 0)$, $Op(C_{ij},d\rightarrow 1)$ and $Op(C_{ij},d\rightarrow 0)$. Let x_i be the i^{th} term, y_j the j^{th} class. Let $(D\text{-}R)^C$ and $(R\text{-}D)^C$ be the class vectors of (D-R) and (R-D), respectively. If A is a vector, let A_i be its i^{th} component. We shall consider every vector as a set in the following sense. If $A_i=1$ and A is a document vector, then $x_i \varepsilon A$. If $A_i=d$ or 0, then $x_i \varepsilon A$. Similarly if $A_j=1$ and A is a class vector, then $y_j \varepsilon A$.

Having defined the above six operations, we shall make use of them as follows. Suppose we are given a (D,R) pair. If the pair satisfies assessment, then there is no need to apply any operation. If D is judged relevant to R but $f(D,R)\leq T$, then an operation or a sequence of operations is applied to the pair so as to increase the value of f(D,R).

Similarly, if f(D,R) T but D is judged irrelevant to R, then another operation or sequence of operations is chosen to decrease f(D,R). Sometimes, a single operation may not be able to bring about any change on a given (D,R) pair. For example, (D-R) has term terms t_1 and t_2 with the class vectors of t_1 and t_2 having a '1' in the j^{th} position. Thus $(D-R)^C_j=1$. Suppose the desired change is to convert $(D-R)^C_j$ from 1 to 0. Then neither the operation $Op(C_{t_1 j},1\rightarrow 0)$ nor the operation $Op(C_{t_2 j},1\rightarrow 0)$ can make $(D-R)^C_j=0$. However $Op(C_{t_1 j},1\rightarrow 0)$ followed by $Op(C_{t_2 j},1\rightarrow 0)$ or the operations in the reverse order convert $(D-R)^C_j$ from 1 to 0. A sequence of operations, $Op(C_{ij},x\rightarrow y)$ with x,y and j fixed while varying i, applied to a (D,R) pair which brings about a desired change in exactly one of $(D-R)^C_j$ and $(R-D)^C_j$ is called a composite operation.

We shall assume that the matching function f is monotonically increasing in the number of class matches and monotonically decreasing in the number of class mismatches. This restriction will be removed later on. One observation we can make is that whether f(D,R) increases, decreases or remains unchanged after an application of an operation or a composite operation is completely independent of the function, f, being chosen.

This observation is stated as follows.

Theorem 4.1. Let f and g be two matching functions. If f_1 and g_1 are the two matching function values of a given (D,R) pair and f_1' and g_1' are their values respectively after an

application of an operation or a composite operation, then

$$f_1 < f_1' \iff g_1 < g_1'$$

$$f_1 = f_1' \iff g_1 = g_1'$$

$$f_1 > f_1' \iff g_1 > g_1'$$

Proof: It is sufficient to prove the theorem for the case of a composite operation. Let the composite operation be $Op(C_{ij}, x \to y)$ for a number of i's. For any given (D,R) pair, the composite operation can at most change the value of $(D\text{-}R)^C_j$ and $(R\text{-}D)^C_j$, i.e., $(D\text{-}R)^C_\ell$ and $(R\text{-}D)^C_\ell$ remain unchanged if $\ell \neq j$. Considering the j^{th} class between $(D\text{-}R)^C_j$ and $(R\text{-}D)^C_j$ before the operation is applied, there are three possibilities, namely, a class match, a no match, and a class mismatch. After the application of the operation, we also have the same three possibilities. Thus, an application of a composite operation produces one and only one of the following nine changes. 1) a class match is converted to a class match; 2) a class mismatch to a class mismatch; 3) a no match to a no match; 4) a mismatch to a class match; 5) a no match to a class match; 6) a mismatch to a no match; 7) a class match to a mismatch; 8) a class match to a no match; 9) a no match to a mismatch. For the first 3 cases, any matching function remains unchanged. For the next three cases, any matching function increases and for the last three cases, any matching function decreases.

Based on this theorem, we shall design an algorithm which is completely independent of the matching function to be chosen.

We now examine whether any arbitrary given (D,R) pair can possibly satisfy a given assessment. If $D \subseteq R$ or $R \subseteq D$, then either (D-R) or (R-D) is empty, which implies that either $(D-R)^C$ or $(R-D)^C$ contains don't care entries only. Suppose $(D-R)^C$ contains don't care entries only, then any $(R-D)^C$ will not give rise to a class match or mismatch. Thus no change in the term classification matrix can alter the value of f(D,R). Let us consider another situation. The highest value which f(D,R) can take on is to have all class matches and no class mismatch. If D is judged relevant to R and f(D,R) takes on its maximal value but $f(D,R) \leq T$, then no change in the classification matrix can make (D,R) satisfy assessment. Similarly if D is judged irrelevant to R and the minimum value of f(D,R) is greater than T, then whatever change in the term classification matrix will not make (D,R) satisfy assessment. Let the highest and lowest values of f(D,R) be f_{HIGH} and f_{LOW} respectively. Based on the above discussion, we obtain the following proposition.

Proposition 4.2. A given (D,R) pair cannot satisfy assessment under any term classification matrix iff one of the following conditions is satisfied

i) (D,R) does not satisfy assessment under some term classification matrix and either $D \subseteq R$ or $R \subseteq D$.

ii) D is relevant to R and $f_{HIGH} \leq T$

iii) D is not relevant to R and $f_{LOW} > T$.

If condition (i) is satisfied, it is likely that the user gives a false assessment of the (D,R) pair. If either condition (ii) or condition (iii) is satisfied, then we should increase the number of classes so that f_{HIGH} can be increased and f_{LOW} can be lowered. Thus, conditions (ii) and (iii) give us an indication of the minimum number of classes to be chosen.

Definition 4.1. A (D,R) pair which cannot possibly satisfy assessment is called a discarded pair.

The next thing we consider is how we should go about increasing or decreasing f(D,R) for a given (D,R) pair. Since decreasing f(D,R) is analogous to increasing it, we shall consider increasing f(D,R) only. Let $(D-R)^C$ and $(R-D)^C$ be $(d_1,d_2,\ldots,d_n)$ and $(r_1,r_2,\ldots,r_n)$ respectively. It is sufficient to examine changing the class vector of (D-R) only, since the other case i.e., changing the class vector of (R-D) is very similar. The first class of $(D-R)^C$ and $(R-D)^C$ is scanned. If $d_1=r_1\neq d$, then no operation applied on the first class can increase f(D,R). If $d_1=r_1=d$, then it takes at least two operations to make the first class a class match. Since later operations applied to other (D,R) pairs may make either d_1 or r_1 a non-d entry, we will postpone making any change for d_1 and r_1. If $d_1\neq r_1$ and we want to use an operation of the form $Op(C_{ij},z\rightarrow y)$ with $x_i\varepsilon D-R$, we have four cases, namely $d_1=0$, $r_1=1$; $d_1=d$, $r_1=1$; $d_1=d$, $r_1=0$; and $d_1=1$, $r_1=0$. (For the other cases, i.e. $(d_1=0,r_1=d)$ and $(d_1=1,r_1=d)$, changing d_1 will not increase f(D,R)). We now consider the case

(d_1=1,r_1=0) in detail. When $Op(C_{i1},1\rightarrow0)$ is applied to all terms i such that $x_i \epsilon$ D-R and C_{i1}=1, $(D-R)_1^C$ will be set to 0, which will then increase the number of class matches by 1. Athough this operation helps the current (D,R) pair to satisfy assessment, it may deteriorate other (D,R) pairs. Thus, we may not want to apply this operation until some other criteria are satisfied. Four strategies will be given later, which will decide whether a given operation should be applied or not. Suppose the operation $Op(C_{i1},1\rightarrow0)$ is decided not applicable, then we should try operation $Op(C_{i1},1\rightarrow d)$. This operation may not improve the (D,R) pair as much as $Op(C_{i1},1\rightarrow0)$ but on the other hand, it has more chance of being decided applicable. Suppose that $Op(C_{i1},1\rightarrow d)$ is decided applicable but the improvement is not sufficient to make (D,R) satisfy assessment, then we should try $Op(C_{i1},d\rightarrow1)$ for some i such that $x_i \epsilon$ D-R and C_{i1}=d. Note that the failure of $Op(C_{i1},1\rightarrow0)$ for <u>all</u> i such that $x_i \epsilon$ D-R and C_{i1}=1 may not necessarily imply the failure of $Op(C_{i1},1\rightarrow d)$ for <u>all</u> i such that $x_i \epsilon$ D-R and C_{i1}=1, succeeded by $Op(C_{i1},d\rightarrow0)$ for <u>some</u> i such that $x_i \epsilon$ D-R and C_{i1}=d. Suppose that the operation is applied but the improvement is still not enough. Then the next class is considered. If all the classes have been considered but the (D,R) pair does not satisfy assessment, then we repeat the process for the class vector of (R-D). If the (D,R) pair still does not satisfy judgement, then the (D,R) pair is stacked and will be handled later on.

Diagrams 4.1 and 4.2 give the flowcharts for increasing and decreasing a given (D,R) pair. Before we proceed further, we will give a detailed example showing how a (D,R) pair is processed.

Example 4.1. Let the matching function be

$$f(D,R) = \cos(D,R) + \frac{4a'-b'}{4a'+b'} \times 0.1$$

where a' and b' are the class matches and mismatches between $(D-R)^C$ and $(R-D)^C$, respectively.

$$D = (0,0,0,0,1,1,1,1)$$

$$R = (0,1,1,0,0,1,1,0) \ .$$

Let the assessment be D not relevant to R. So, we want $f(D,R) \leq$ the threshold T=0.46.

The term classification matrix, C, before this (D,R) pair is processed is

$$C = \underline{\text{terms}} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \overset{\underline{\text{classes}}}{\begin{bmatrix} 1 & 1 & d & 0 & d \\ d & 0 & 0 & 1 & 0 \\ 0 & d & 0 & d & 1 \\ d & d & d & 0 & 1 \\ 0 & 0 & d & d & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & d & 0 \\ 1 & d & d & d & 0 \end{bmatrix}} \ .$$

$$(D-R) = (0,0,0,0,1,0,0,1)$$

$$(D-R)^C = (0,0,d,d,1) \cup (1,d,d,d,0)$$

$$= (1,0,d,d,1)$$

To increase f(D,R)

Let $(D-R)^C = (d_1, d_2, \ldots, d_n)$ and $(R-D)^C = (r_1, r_2, \ldots, r_n)$

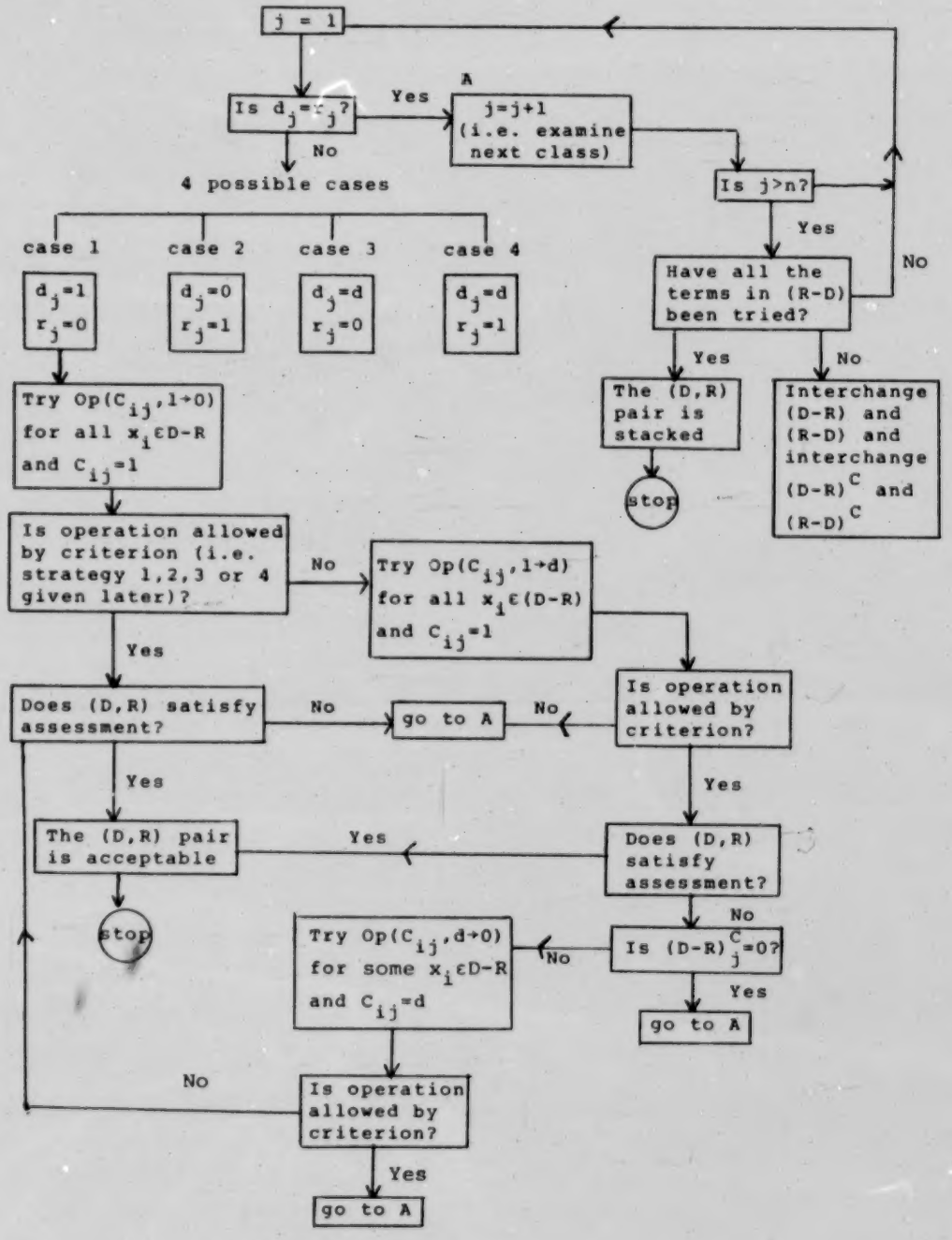

Diagram 4.1. (The other cases are similar to case 1).

BEST COPY AVAILABLE

To decrease $f(D,R)$

Let $(D-R)^C = (d_1, d_2, d_3, \ldots, d_n)$ and $(R-D)^C = (r_1, r_2, \ldots, r_n)$

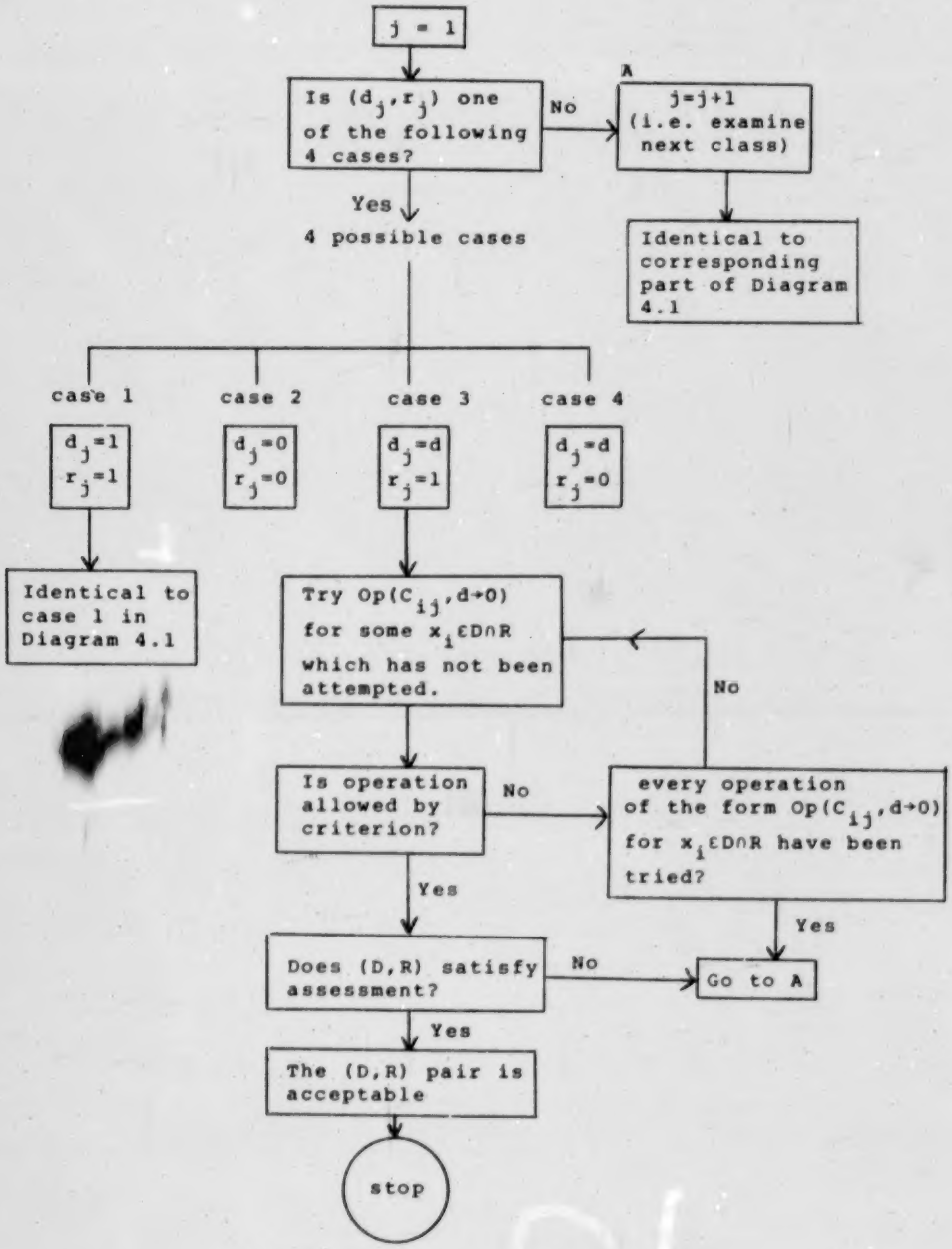

Diagram 4.2. (Case 3 is done in detail)

$$(R-D) = (0,1,1,0,0,0,0,0)$$
$$(R-D)^C = (d,0,0,1,0) \cup (0,d,0,d,1)$$
$$= (0,0,0,1,1)$$

From $(D-R)^C$ and $(R-D)^C$, we have a'=2 and b'=1

$$f(D,R) = \cos(D,R) + \frac{4.2-1}{4.2+1} \times 0.1$$
$$= 0.5 + .0778 = 0.5778$$

Since $f(D,R)>T$, we want to decrease $f(D,R)$. There is already a class mismatch in the first class. Thus we scan the second class. We would like to change the second class of $(D-R)^C$ to a 1 so that we can have a mismatch in the second class. Thus the class chosen is 2. The term chosen is the first non-zero term of (D-R) whose second class equals 0 or d, i.e. term 5. Thus the operation is $Op(C_{52},0\rightarrow 1)$ or $Op(C_{52},d\rightarrow 1)$.

Suppose that the operation is not allowed for not satisfying certain criteria. Then the next term chosen is term 8 of (D-R). Thus the operation is $Op(C_{82},d\rightarrow 1)$, since the second class of term 8 is d.

Suppose that the operation is again not allowed. Then we would like to change the second class of $(D-R)^C$ to a d so that we will not have a class match in the second class. The term we choose is the first non-zero term of (D-R) whose class equals 0. Thus the operation is $Op(C_{52},0\rightarrow d)$.

Suppose the operation is allowed by the strategy. The operation is applied and the new classification matrix becomes

Classes

$$\underline{\text{terms}} \begin{bmatrix} 1 & 1 & d & 0 & d \\ d & 0 & 0 & 1 & 0 \\ 0 & d & 0 & d & 1 \\ d & d & d & 0 & 1 \\ 0 & d & d & d & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & d & 0 \\ 1 & d & d & d & 0 \end{bmatrix} .$$

Now, a' = 1 and b' = 1.

The new f(D,R) = 0.5600 and we still have to decrease f(D,R).

The other operations attempted are listed below:

$Op(C_{53}, d \to 1)$ which is not allowed;

$Op(C_{83}, d \to 1)$ which is not allowed;

$Op(C_{54}, d \to 0)$ which is allowed.

At this stage, f(D,R) = 0.5333 and we still have to decrease f(D,R). The next operation is

$Op(C_{55}, 1 \to 0)$ which is not allowed.

The next operation is $Op(C_{55}, 1 \to d)$. Since this operation fails, there is no need to try $Op(C_{85}, 0 \to d)$. After this operation, we interchange (D-R) and (R-D) and also $(D-R)^C$ and $(R-D)^C$ and other operations are tried.

Now, we present the four strategies which decide whether a given operation is applicable or not.

Definition 4.2. A (D,R) pair responds favorably to an operation or a composite operation if one of the following conditions is satisfied

i) if D is relevant to R, then one of the following must be satisfied

   a) the number of class mismatches is decreased and there is no decrease in the number of class matches

   b) the number of class matches is increased and there is no increase in the number of class mismatches.

ii) if D is not relevant to R, then one of the following must be satisfied

   a) the number of class matches is decreased and there is no decrease in the number of class mismatches

   b) the number of class mismatches increases and there is no increase in the number of class matches.

iii) there is no change in the number of class matches or mismatches.

A (D,R) pair responds strictly favorably to an operation or a composite operation if either (i) or (ii) is satisfied.

Definition 4.3. An acceptable (D,R) pair is one which satisfies assessment under the current term classification matrix.

A stacked (D,R) pair is one which does not satisfy assessment under the current term classification but may

possibly satisfy assessment under some other classification matrix.

Each of the strategies which we will present ensures that any operation, if applicable, helps the present (D,R) pair in satisfying assessment but also guarantees that the previously processed (D,R) pairs behave "reasonably well". Hopefully more and more (D,R) pairs are brought into assessment as more and more operations are performed.

Strategy 1. All acceptable and stacked pairs which were processed previously respond favorably and the present (D,R) pair responds strictly favorably.

Definition 4.4. A (D,R) pair is undisturbed by an operation if the operation does not change the pair which satisfies assessment into one which does not.

Thus, a (D,R) which does not satisfy assessment is undisturbed by any operation. While strategy 1 does not allow any deterioration of any (D,R) pair which can satisfy assessment, strategy 2 will certain deterioration, provided it is not "too bad".

Strategy 2. All acceptable (D,R) pairs which were previously processed are undisturbed and the current (D,R) pair responds strictly favorably.

Definition 4.5. The number of favorable changes on a (D,R) pair is counted in the following way.

i) if D is relevant to R, then each of the following changes is counted as one favorable change
   a) there is no change in the number of class matches and the number of class mismatches decreases by 1.
   b) there is no change in the number of class mismatches and the number of class matches increases by 1.

ii) if D is not relevant to R, then each of the following changes is counted as a favorable change
   a) there is no change in the number of class matches and the number of class mismatches increases by 1.
   b) there is no change in the number of class mismatches, and the number of class matches decreases by 1.

Thus, if D is relevant to R and a mismatch is converted to a match, then the number of favorable changes is 2.

The number of unfavorable changes on a (D,R) pair is counted as above except that the words 'relevant' and 'not relevant' are interchanged in (i) and (ii).

The number of favorable changes on a set of (D,R) pairs is given by the sum of all favorable changes on each (D,R) pair in the set. The number of unfavorable changes on a set

of (D,R) pairs is calculated similarly.

Strategy 3. The number of favorable changes on all acceptable and stacked pairs is greater than the number of unfavorable changes on all acceptable and stacked pairs.

While strategy 2 does not allow any (D,R) pair which satisfies assessment to deteriorate "too badly", strategy 3 allows deterioration of any (D,R) pair to any degree provided that the set of all (D,R) pairs which can possibly satisfy assessment "improves" on the whole. Generally, this will improve system performance as measured by recall and precision effectively. When a (D,R) pair is judged to be relevant (irrelevant) and all documents which rank higher (lower) than D with respect to R are relevant (irrelevant) to R, increasing (decreasing) the rank of D any further will not improve recall and precision. Thus a better strategy than strategy 3 would be to count a change on a (D,R) pair as favorable only if there is at least one irrelevant (relevant) document whose rank is higher (lower) than D and it is possible for D to pass such a document in ranking. However, any method attempting to implement that would involve sorting the correlation coefficients of documents with respect to each request and a lot of computing time is required. To avoid it, we use the following definition in specifying strategy 4.

Definition 4.6. A favorable change in a restricted domain on a (D,R) pair is a favorable change which satisfies one of the following conditions.

i) if D is relevant to R, $f(D,R)<T_2$ before the operation is applied.

ii) if D is not relevant to R, $f(D,R)>T_1$ before the operation is applied.

T_2 and T_1 are constants ($T_2>T_1$) specifying the restricted domain we are interested in. Any document whose correlation coefficient with R is greater (less) than T_2 (T_1) is assumed to be relevant (irrelevant) to R. Thus, increasing (decreasing) f(D,R) beyond T_2 (T_1) would not improve recall and precision.

Strategy 4 The number of favorable changes in restricted domain on all acceptable and stacked pairs is greater than the number of unfavorable changes on all acceptable and stacked pairs.

Diagram 4.3 illustrates how all the (D,R) pairs are processed in the first cycle. At the end of the first cycle, there are 3 sets of (D,R) pairs, a set containing acceptable (D,R) pairs, one containing stacked pairs and the last one containing discarded pairs. In the second and subsequent cycles, attempts are made to change stacked (D,R) pairs into acceptable ones by going through the same flowchart but processing the set of stacked pairs only. However, when a strategy is applied, all pairs except those discarded are taken into consideration. The stopping criteria for the

BEST COPY AVAILABLE

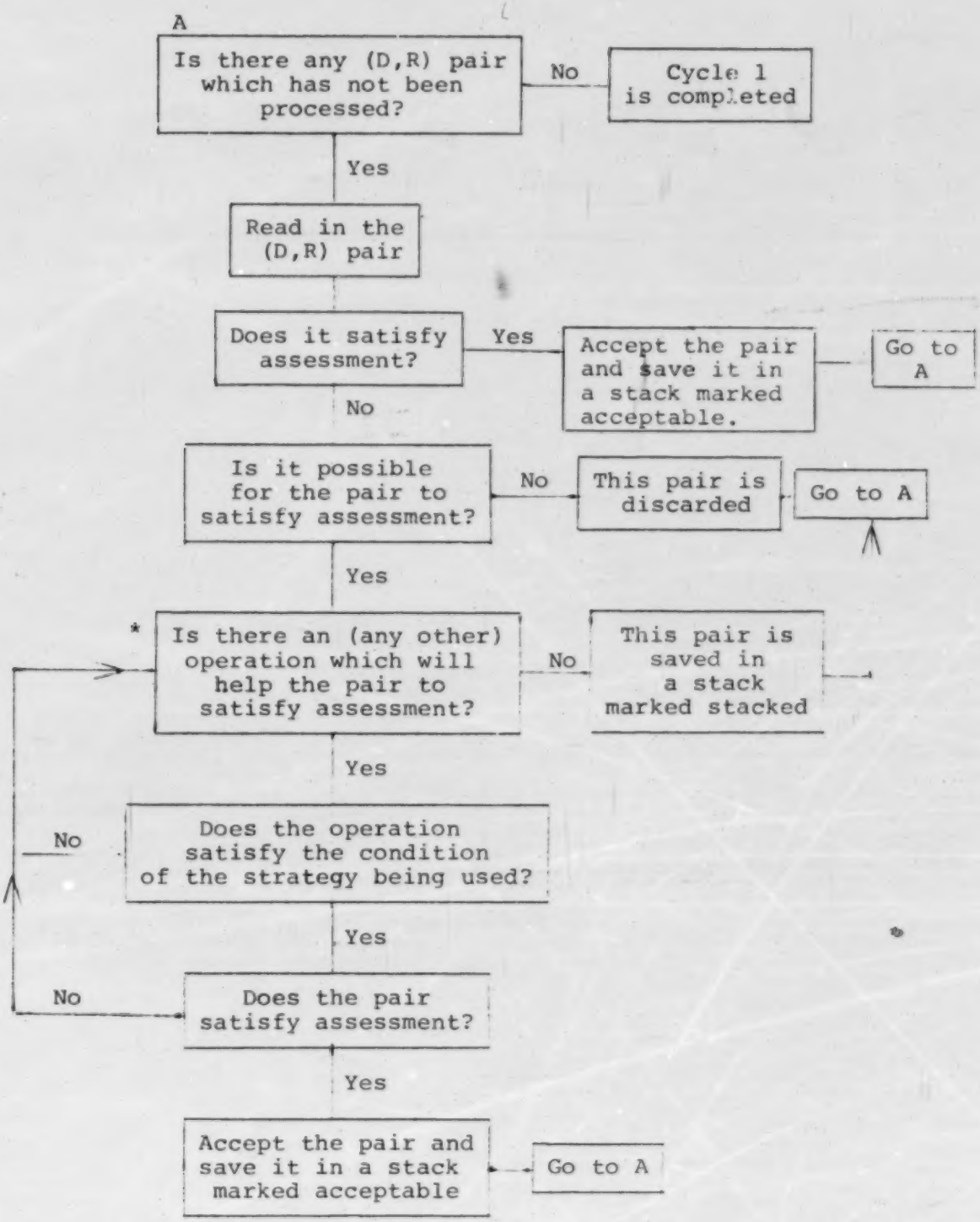
A
Is there any (D,R) pair which has not been processed?
No
Cycle 1 is completed
Yes
Read in the (D,R) pair
Does it satisfy assessment?
Yes
Accept the pair and save it in a stack marked acceptable.
Go to A
No
Is it possible for the pair to satisfy assessment?
No
This pair is discarded
Go to A
Yes
Is there an (any other) operation which will help the pair to satisfy assessment?
No
This pair is saved in a stack marked stacked
Yes
No
Does the operation satisfy the condition of the strategy being used?
Yes
No
Does the pair satisfy assessment?
Yes
Accept the pair and save it in a stack marked acceptable
Go to A

Diagram 4.3. Main program for cycle 1.

strategies are as follows. For strategies 1, 3 and 4, the algorithm halts iff no operation is made between the last cycle and the present one or there is no stacked pair. For strategy 2, the algorithm halts iff there is no increase in the number of acceptable pairs between the last cycle and the present one or there is no stacked pair.

We will show that all the strategies converge.

Proposition 4.3. Strategy 1 converges in a finite number of steps.

Proof: Let f = the number of favorable changes in all acceptable and stacked pairs. f is bounded above by the number of (D,R) pairs times the maximum number of favorable changes a (D,R) pair can possibly have. f is monotonically increasing in the number of operations made because each operation which is applied must increase f by at least 1. A monotonic increasing function which takes on integer value and is bounded above must converge in finite number of steps.

Proposition 4.4. Let f = the number of acceptable (D,R) pairs. Since the number of acceptable (D,R) pairs is less than or equal to the number of (D,R) pairs, f is bounded above by the number of (D,R) pairs. By the definition of strategy 2, f either increases or it remains the same from the i^{th} cycle to the $(i+1)^{st}$ cycle. In the latter case, strategy 2 stops. For the former case, strategy 2 stops by an argument similar to that of Proposition 4.3.

Proposition 4.5. Strategy 3 (4) stops in a finite number of steps.

Proof: Let f = the number of favorable changes (in restricted domain) on all acceptable and stacked (D,R) pairs - the number of unfavorable changes on the same pairs. Then apply an argument similar to that of Proposition 4.3.

We are now in a position to modify the algorithm whose flowchart is described by diagram 4.3 if the matching function f is non-decreasing in the number of class matches and non-increasing in the number of class mismatches. If f were a monotonic function, then an operation which will help a (D,R) pair to satisfy assessment would be one which increases the number of class matches or decreases the number of class mismatches in the case D is relevant to R and which either decreases the number of class matches or increases the number of class mismatches in the case D is not relevant to R. If f is only non-decreasing in the number of class matches and non-increasing in the number of class mismatches, then in order to decide whether an operation really helps a (D,R) pair to satisfy assessment, we not only have to check the above conditions but also compute the value of f and see whether any change has been made. This is the only change we have to make in the algorithm (see the box marked with * in diagram 4.3).

5. Experimental Results

In this section, we shall compare the performance of the four strategies with the performance of a matching function which does not make use of the concept of class matches and mismatches. The function chosen is the cosine function used in the SMART Project [6]. In Diagram 5.2, this function is referred to as f_1. The matching function which uses the concept of class matches and mismatches is $f_2(D,R) = \cos(D,R) + \frac{k_1a'-b'}{k_1a'+b'} k_2$ where a' and b' are the number of class matches and mismatches, respectively and k_1 and k_2 are constants. The comparison of the two matching functions will be carried out in two ways: the percentage improvement of the new function using a particular strategy over the cosine function in terms of the number of (D,R) pairs satisfying assessment and in terms of precision and recall.

In Diagram 5.2, the results are shown with $k_1 = 4$ and $k_2 = 0.1$. In Diagram 5.1, the documents and the request pairs, the assessment of the pairs and the threshold are given.

The percentage improvement of a particular strategy over f_1 is given by the formula {(no. of (D,R) pairs satisfying assessment using that strategy) - (no. of (D,R) pairs satisfying assessment using f_1)}÷{no. of (D,R) pairs satisfying assessment using f_1)}×100%. The number of (D,R) pairs satisfying assessment by f_1 and by the other strategies is shown in Diagram 5.4.

Example:

D_1 = (0,1,1,0,1,0,0,1) R_1 = (0,0,1,0,1,0,0,1)

D_2 = (0,0,0,0,1,1,1,1) R_2 = (1,0,1,1,0,0,1,1)

D_3 = (0,1,0,1,0,1,0,1) R_3 = (0,1,1,0,0,1,1,0)

D_4 = (1,0,1,0,1,0,1,0)

D_5 = (1,1,1,1,0,0,0,0)

D_6 = (0,0,1,1,0,0,1,1)

D_7 = (1,1,0,0,1,1,0,0)

D_8 = (1,1,1,0,0,1,0,1)

D_9 = (0,1,0,0,1,0,1,1)

D_{10} = (0,1,0,1,1,0,1,0)

The number of terms = 8 , the number of classes = 5

Threshold T = 0.46 .

Assessment matrix Z = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

Diagram 5.1.

Number of (D,R) pairs given = 30

Number of (D,R) pairs discarded using f_2 is 6

i.e., there are 6 (D,R) pairs which can never satisfy assessment using f_2 with the parameters $k_1 = 4$ and $k_2 = 0.1$.

Number of (D,R) pairs satisfying assessment by				
f_1	f_2 with strategy 1	f_2 with strategy 2	f_2 with strategy 3	f_2 with strategy 4
16	17	22	21	20

Initial term classification matrix C

$$\underline{\text{terms}} \begin{bmatrix} 0 & 1 & d & 0 & d \\ d & 0 & d & 1 & 0 \\ 0 & d & 0 & d & 1 \\ d & d & d & 0 & 1 \\ 0 & 0 & 1 & d & d \\ d & d & 1 & 1 & 0 \\ 1 & 0 & 1 & d & 0 \\ 1 & d & d & d & 0 \end{bmatrix}$$

$\underline{\text{classes}}$

f_1 = cosine function

$$f_2(D,R) = \cos(D,R) + \left(\frac{k_1 a' - b'}{k_1 a' + b'} \right) k_2$$

Diagram 5.2.

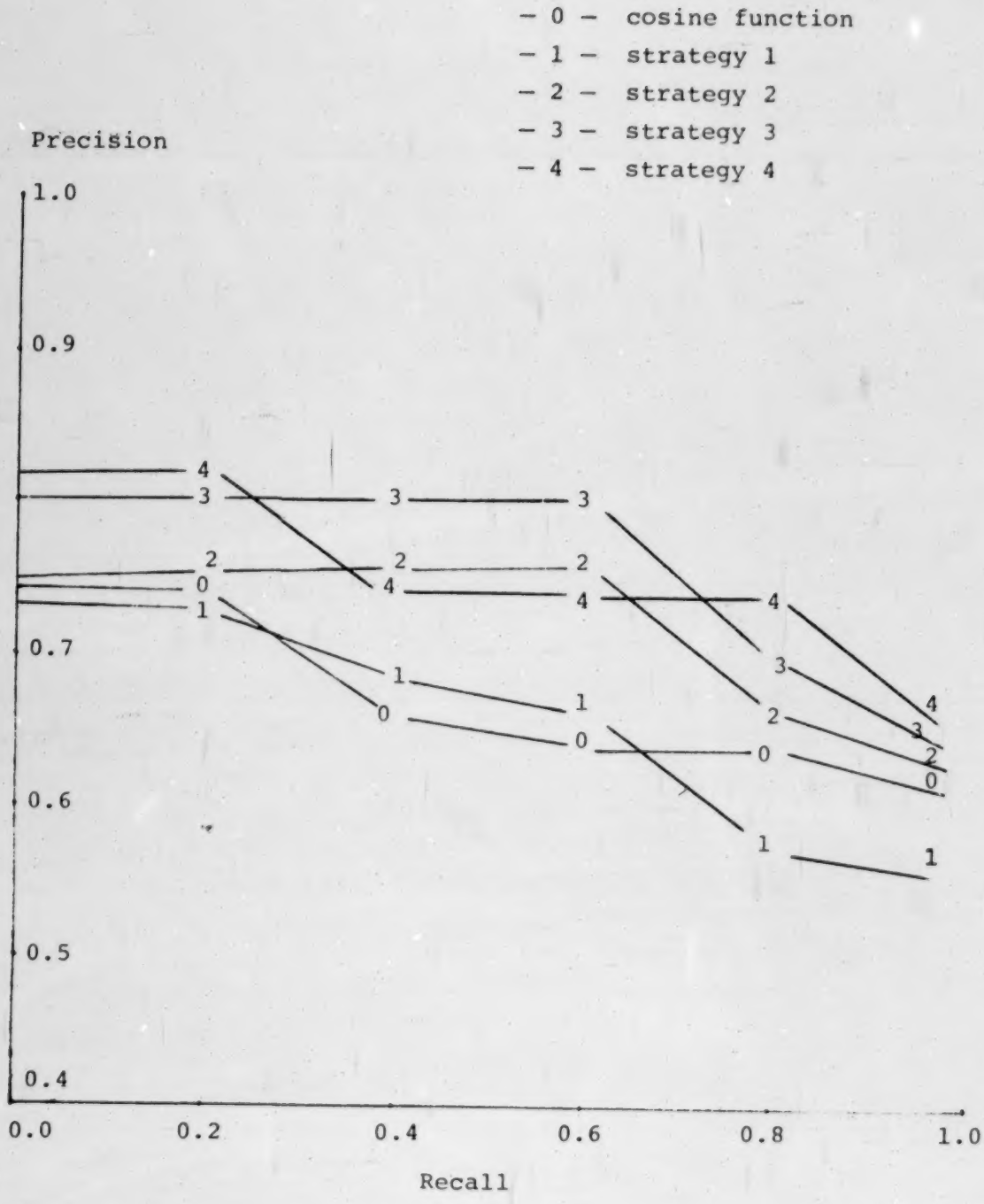
– 0 – cosine function
– 1 – strategy 1
– 2 – strategy 2
– 3 – strategy 3
– 4 – strategy 4
Precision
1.0
0.9
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0
Recall

Diagram 5.3.

Percentage improvement of strategy i over the cosine function in terms of the number of (D,R) pairs satisfying assessment			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
6.25	37.5	31.25	25

Diagram 5.4.

Percentage improvement of strategy i over the cosine function in terms of Recall and Precision			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
-4.3	6.6	13.1	13.1

Diagram 5.5.

The percentage improvement of the strategies over f_1 is given by Diagram 5.5. The precision and recall curves for the strategies and the cosine function are plotted in Diagram 5.3.

There are a number of parameters that we can vary in performing the above experiments. The first variation involves changing the initial term classification matrix. The results are shown in Diagrams 5.6 and 5.7. The second parameter varied is the order in which the (D,R) pairs are processed. Diagrams 5.8 and 5.9 show the results when all the (D,R) pairs are processed in reverse order. Next, we vary the constants k_1 and k_2. From the above (D,R) pairs, we find the highest value which cosine (D,R) can have among all (D,R) pairs is .6708 when both (D-R) and (R-D) are non-empty. If the highest value which $f_2(D,R)$ is allowed to reach is 1, then we should choose $k_2 = 1 - 0.6708 = .3292$. Since the number of classes equals 5, using $k_1 = 4$ would imply that if there is a match, then $f_2(D,R) \geq f_1(D,R)$. Thus, we should lower k_1 to 2 if we want class mismatches to have some importance as compared to class matches. The results of using $k_2 = .3292$ and $k_1 = 2$ are shown in Diagrams 5.10 and 5.11. The computing time for the four strategies in the above experiments is shown in Diagram 5.12.

By looking at the experimental results, strategy 2 is far superior to strategy 1 in terms of the number of (D,R) pairs satisfying assessment. This is not at all surprising

Percentage improvement of strategy i over the cosine function in terms of the number of (D,R) pairs satisfying assessment			
Strategy 1	Strategy 2	Stratgey 3	Strategy 4
25	31.25	31.25	31.25

A different initial term classification matrix is used

Diagram 5.6.

Percentage improvement of strategy i over the cosine function in terms of Recall and Precision			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
4.3	10.3	13.3	11.7

A different initial term classification matrix is used

Diagram 5.7.

Percentage improvement of strategy i over the cosine function in terms of the number of (D,R) pairs satisfying assessment			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
6.25	25	12.50	31.25

The (D,R) pairs are processed in reverse order

Diagram 5.8.

Percentage improvement of strategy i over the cosine function in terms of Recall and Precision			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
6.6	-0.9	16.8	16.8

The (D,R) pairs are processed in reverse order

Diagram 5.9.

Percentage improvement of strategy i over the cosine function in terms of the number of (D,R) pairs satisfying assessment			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
43.75	62.5	50	56.25

Constants $k_1 = 2$, $k_2 = .3292$

Diagram 5.10.

Percentage improvement of strategy i over the cosine function in terms of Recall and Precision			
Strategy 1	Strategy 2	Strategy 3	Strategy 4
12.5	22.0	22.6	22.5

Constants $k_1 = 2$, $k_2 = .3292$

Diagram 5.11.

Computing time of the different strategies in the above experiments				
	Strategy 1	Strategy 2	Strategy 3	Strategy 4
Original matrix	23.70 sec.	32.23 sec.	48.75 sec.	39.53 sec.
Altered matrix	13.93 sec.	12.83 sec.	22.24 sec.	27.07 sec.
(D,R) pairs processed in reverse order	17.35 sec.	14.89 sec.	39.09 sec.	43.42 sec.
Changing k_1 , k_2 to $k_1 = 2$, $k_2 = .3292$	45 sec.	30.17 sec.	77.96 sec.	77.81 sec.

Diagram 5.12.

because any operation that is allowed in strategy 1 is also allowed in strategy 2 under the same term classification matrix, and strategy 1 is restricting too many operations to be applicable. In the experiments that we ran, strategy 2 is even better than strategy 3 in the number of (D,R) pairs satisfying assessment, but strategy 3 is better than strategy 2 in terms of recall and precision. Strategy 3 guarantees that any operation that is applied improves the global condition of all the (D,R) pairs while strategy 2 is more interested in the satisfiability of the current (D,R) pair. Strategy 2 allows an operation which deteriorates some of the (D,R) pairs that have been processed, provided that the deterioration in each pair is not "too much". The operation may hurt the global behavior of the (D,R) pairs and thus on the whole, we expect strategy 3 to be superior in recall and precision. Strategy 4's performance should be better than that of strategy 3 but it is not easy to fix the values of the thresholds T_1 and T_2 to obtain good results.

When the parameter k_2 is changed from 0.1 to 0.3292, most of the (D,R) pairs which were discarded when $k_2 = 0.1$ become available for processing. Thus, the performance of all the strategies improves significantly. On the other hand, a lot more operations have to be performed and the computing time increases considerably. When the other parameters are changed, i.e., changing the initial term

classification and the order in which the (D,R) pairs are processed, strategy 2 still performs very well in the number of (D,R) pairs satisfying assessment, and strategies 3 and 4's performance remains well in terms of recall and precision. When a different initial term classification matrix is used, the performance of strategy 1 improves from 6.25% to 25% in terms of the number of (D,R) pairs satisfying assessment.

In general, we believe that the strategies which are designed for a particular purpose (strategy 2 maximizing the number of (D,R) pairs satisfying assessment and strategies 3 and 4 improving recall and precision) are rather independent of the order in which the (D,R) pairs are processed and also of the initial term classification matrix. (A particular initial term classification matrix may make one strategy perform a lot better and another one a lot worse. But a randomly generated one with mostly don't cares as entries will give approximately the same result each time.) On the other hand, the performance of strategy 1 is highly dependent on the above parameters.

We shall try strategy 3 on a subset of the ADI collection which has 82 documents and 35 requests. The first five requests are chosen and the (D,R) pairs are chosen satisfying one of the following conditions (i) D is relevant to R or (ii) D ranks within the top twenty documents in correlating with R. The number of (D,R) pairs satisfying the above criterion

equals 103. The number of terms and classes used is 1187 and 20, respectively. The results are shown in Diagram 5.13. The chosen threshold =0.25 with constants k_1 = 3 and k_2 = 0.1. The initial term classification matrix is too large to be given here. The total number of (D,R) pairs is 103 and strategy 3 accepts 101 pairs and 2 pairs are discarded (with no stacked pairs). The computing time spent is 44 minutes, but the improvement obtained is unexpectedly good. It has a 110.9% improvement over the cosine function.

6. Computation Time of the Heuristic Methods.

In this section, we shall give a rough estimation of the computing time of strategy 3. In the experiments shown in the last section, we saw that strategy 3 usually takes more time than the other strategies. Thus a rough bound for strategy 3 would serve as a bound for other strategies.

Let the number of (D,R) pairs be n. Let the average number of terms in a document be p and that in a request be q. Suppose the number of classes is r.

If D is relevant to R, then the number of favorable changes in one class is at most 2, namely changing a mismatch to a no match and then from a no match to a class match. Similarly if D is not relevant to R, there are at most 2 favorable changes per class. Thus the total number of favorable changes for a (D,R) pair is at most 2r, implying that the total number of favorable changes for all (D,R) pairs is bounded by 2rn. From the i^{th} cycle to the $(i+1)^{st}$ cycle,

BEST COPY AVAILABLE

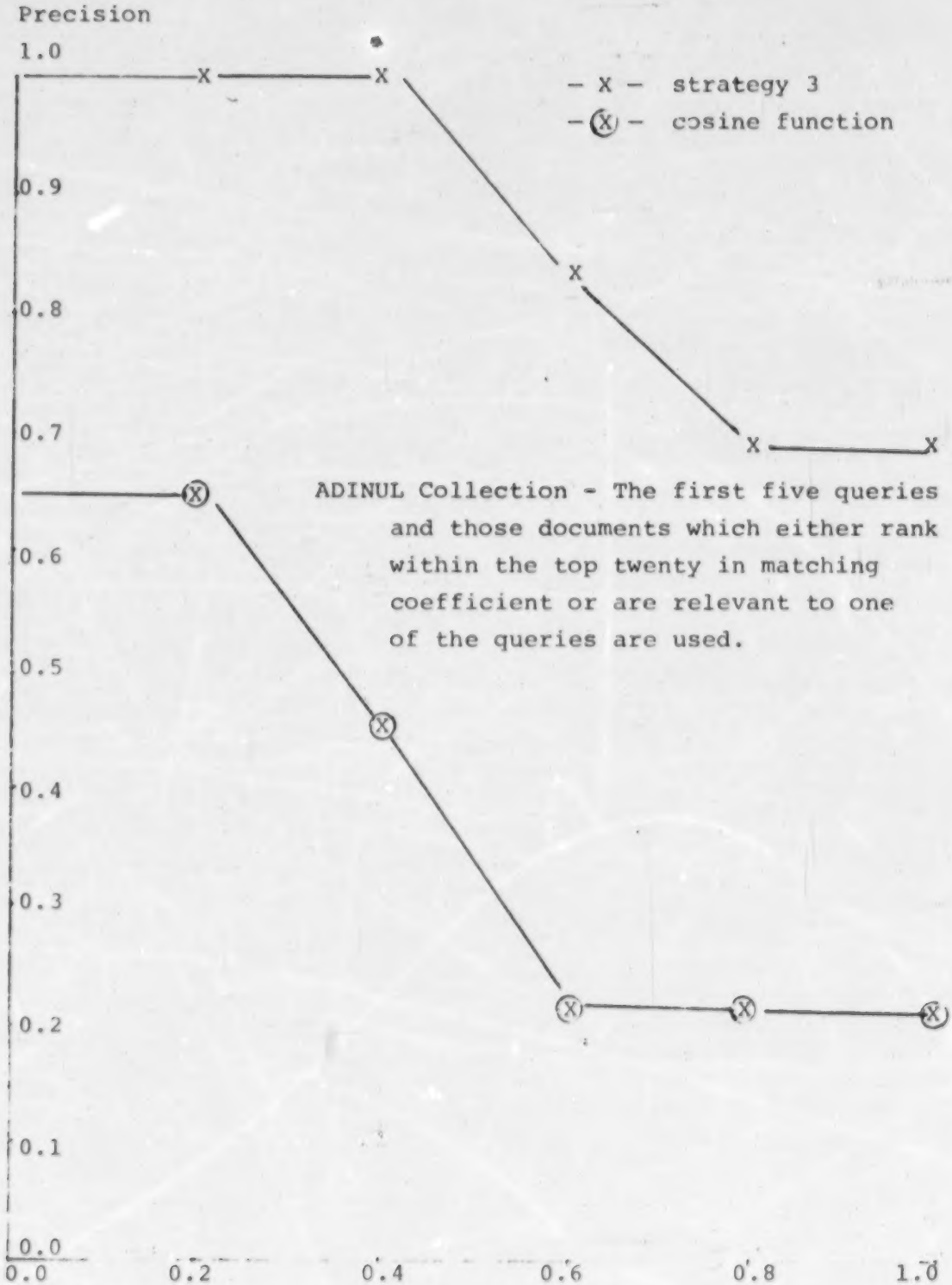

Diagram 5.13.

at least one operation is applied. Therefore, the number of favorable changes increases by at least one. So, the number of cycles is at most 2rn. Now in each cycle, all the stack pairs are processed and the number of stack pairs is less than or equal to n. For each pair, the number of operations that may be favorable (but not necessarily applied) is at most 2(p+q)r, because there are at most 2 favorable changes for each class and each favorable change may be caused by any one of the (p+q) terms (the cardinality of $(D-R) \cup (R-D)$ is bounded by (p+q)). For each operation which may be applicable, we have to go through all acceptable and stacked pairs, which is not more than n. If (D-R), (R-D), $(D-R)^C$ and $(R-D)^C$ are stored, then checking whether a given operation is favorable to a pair or not takes constant time. Thus the total time taken is

[2rn]	[n]	[2(p+q)r]	[n]	$= O(n^3 r^2 (p+q))$
number of cycles	no. of (D,R) pairs processed per cycle	no. of operations considered for each (D,R) pair	no. of (D,R) pairs to be checked to decide whether an operation is applicable or not	

This is really a large number when the number of (D,R) pairs is large or the number of classes is large. But this number does not represent the actual running time of strategy 3. In the first cycle, the computing time is $O(n^2 r(p+q))$.

For each cycle after the first cycle, the number of stacked pairs is extremely small and for practical purposes, can be assumed to be bounded by a constant. Thus the amount of computer time spent per cycle after the first cycle is $O((p+q)rn)$. In all the experiments performed, the number of cycles was at most 4 and in many cases, the number went down to 2. A more realistic situation is to assume that the number of cycles is a constant. Under that assumption, the computing time spent is $O(n^2r(p+q)) + O((p+q)rn) = O(n^2(p+q)r)$. The computer times used in the different experiments confirm the above calculation.

7. Conclusion

The formal construction of term classes is shown to be a difficult process computationally. The approximation to the construction by the heuristic methods, especially the third strategy, is verified by experimental results to be a sufficiently close one. However, the computer time required by the heuristic methods is still too large for any real document collection. Future research should be directed at getting better heuristic algorithms. Our next set of experiments will be aimed at finding out whether the construction obtained by a given set of queries and user's assessment agrees with that obtained by a different set.

Acknowledgements

I would like to express my gratitude to Professor D.M. Jackson who suggested the problem and gave me many useful suggestions. Thanks are also due to Professor S.K. Sahni who co-operated with me in parts of this paper and R. Murton who programmed the algorithms.

APPENDIX

[The results of this appendix were obtained jointly with S.K. Sahni.]

Theorem 1. For every formula in conjunctive normal form having exactly three literals per clause, there exists a set of documents D's, a set of requests R's, a matching function f, a threshold T, an assessment matrix Z and a partial solution for the satisfiable assessment problem such that the formula is satisfiable <=> all (D,R) pairs are satisfied.

Proof: Let P be a formula in conjunctive normal form with n variables $x_1, x_2, \ldots, x_n$ and m clauses $c_1, c_2, \ldots, c_m$. Thus P is of the form $\bigwedge_{i=1}^{m} c_i$. Further, each clause c_i has exactly three literals and is of the form $c_{i1} \vee c_{i2} \vee c_{i3}$ where c_{ij} is either a variable or its complement. We shall show how to obtain a set of (D,R) pairs, a matching function f, a threshold T, an assessment matrix Z and a partial

solution such that all (D,R) pairs are satisfied iff the formula P is satisfiable.

Corresponding to each clause c_i there will be one R_i and one D_i. We shall have 3n +m distirct terms:

$\{g_j\}_{j=1}^{m}$; $\{t_j\}_{j=1}^{n}$; $\{z_j\}_{j=1}^{n}$ and $\{\bar{z}_j\}_{j=1}^{n}$.

For each clause c_i construct D_i and R_i as follows. R_i and D_i have four terms each. The terms of D_i are given by

i) g_i is in D_i

ii) z_i is in D_i iff the literal x_i occurs in c_i.

iii) $\bar{z}_i$ is in D_i iff the literal $\bar{x}_i$ occurs in c_i.

The terms of R_i are given by

i) g_i is in D_i

ii) t_i is in D_i iff the variable x_i is in clause c_i (variable x_i occurs in c_i iff either the literal x_i or the literal $\bar{x}_i$ occurs in c_i).

Each class vector has (n + 1) components and let the partial solution be specified as follows.

i) for the terms $g_1, g_2, \ldots, g_m$, the class vector for each such term is $(d_1, d_2, \ldots, d_n, 1)$ where each d_i is d.

ii) for the terms $t_1, t_2, \ldots, t_n$, the class vector for a term t_i is $(d_1, d_2, \ldots, d_{i-1}, y_i, d_{i+1}, \ldots, d_{n+1})$ for $1 \le i \le n$ where y_i is to be assigned 0 or 1 and each d_i is d.

iii) for the terms $z_1, z_2, \ldots, z_n$, the class vector for z_i is

$(d_1, d_2, \ldots, d_{i-1}, 1, d_{i+1}, \ldots, d_{n+1})$.

iv) for the terms $\bar{z}_1, z_2, \ldots, z_n$, the class vector for $\bar{z}_i$ is

$(d_1, d_2, \ldots, d_{i-1}, 0, d_{i+1}, \ldots, d_{n+1})$.

The y_i as appeared in (ii) above corresponds to the variable x_i of formula P, i.e., $x_i = y_i$, $1 \leq i \leq n$.

The matching function f is given by $f(D_i, R_j) = 4a' - b'$ where a' and b' are the class matches and mismatches between the class vectors of $(D_i - R_j)$ and $(R_j - D_i)$, respectively. The threshold T is 0 and the entries of the assessment matrix are all 1's.

We will first shown that all (D_i, R_j) pairs with $i \neq j$ satisfy assessment. For $i \neq j$, $g_i \varepsilon (D_i - R_j)$ and $g_j \varepsilon (R_j - D_i)$. Thus the class vector of $(D_i - R_j) = (X, X, X, \ldots, X, 1)$ where X can be 0, 1 or d but the number of non-d entries among the X's is 3. This is because each clause has exactly 3 literals and each term which corresponds to a literal has exactly 1 non-d entry by the above construction. Similarly the class vector of $(R_j - D_i) = (X, X, X, \ldots, X, 1)$, with the same condition on X. There is at least one class match between $(R_j - D_i)^C$ and $(D_i - R_j)^C$ because there is already a class match in the $(n+1)^{st}$ position. Since the number of non-d entries in

each class vector is 4 (including the 1 in the $(n+1)^{st}$ position), the number of class mismatches ≤ 3. By the matching function f given above, $f(D_i, R_j) \geq 1 \cdot 0$ and thus satisfies assessment.

For $i=j=\ell$, we are considering the document request pair (D_ℓ, R_ℓ). By the construction above, $(D_i - R_i)^C = (\delta_1, \delta_2, \ldots, \delta_{n+1})$ where

$$\delta_p = \begin{cases} 1 \text{ if literal } x_p \in \text{ clause } c_\ell \\ 0 \text{ if literal } \bar{x}_p \in \text{ clause } c_\ell \\ d \text{ otherwise.} \end{cases}$$

and there are exactly 3 non-d entries among the δ's.

Similarly, $(R_i - D_i)^C = (w_1, w_2, \ldots, w_{n+1})$ where

$$w_p = \begin{cases} y_p \quad \text{if } x_p \text{ or } \bar{x}_p \in \text{ clause } c_\ell \\ \\ d \text{ otherwise.} \end{cases}$$

and there are exactly 3 non-d entries among the w's.

The positions in which the y's occur in $(R_i - D_i)^C$ are exactly where the non-d entries in $(D_i - R_i)^C$ occur. Thus if $c_i = (X_1 \vee X_2 \vee \bar{X}_3)$, then

$$(D_i - R_i)^C = (1, 1, 0, d, \ldots, d)$$

and

$$(R_i - D_i)^C = (y_1, y_2, y_3, d, \ldots, d).$$

If the above clause is satisfiable, then $x_1 = 1$ or

$x_2 = 1$ or $x_3 = 0$

$\Rightarrow y_1 = 1$ or $y_2 = 1$ or $y_3 = 0$

$\Leftrightarrow$ there is at least a class match between $(D_i - R_i)^C$ and $(R_i - D_i)^C$

$\Leftrightarrow f(D_i, R_i) \geq 1 > 0$

$\Leftrightarrow$ the (D,R) pair satisfies assessment.

Since this is true for all clauses and all $\{(D_i, R_i)\}_{i=1}^{m}$ pairs and all $\{(D_i, R_j)\}_{i \neq j}$ pairs satisfy assessment, the formula is satisfiable iff all (D,R) pairs satisfy assessment.

Note that the above construction can be done in deterministic polynomial time.

Lemma 2: If the satisfiable problem with a partial solution can be done in deterministic polynomial time, then we can decide if all the (D,R) pairs can be satisfied.

Proof: Let the number of (D,R) pairs be k. The maximum number of (D,R) pairs satisfying assessment =k iff all (D,R) pairs can be satisfied. Therefore, deciding whether all (D,R) pairs can be satisfied involves comparing the maximum number of (D,R) pairs satisfying assessment with k.

Theorem 3: If there is a deterministic polynomial algorithm for the exactly 3 literal satisfiability problem, then there is one for the satisfiable assessment problem with a partial solution.

Proof: If there is a deterministic polynomial algorithm for the exactly 3 literal satisfiability problem, then any non-deterministic turing machine running in polynomial time can

be simulated by a deterministic turing machine in polynomial time (see [5]).

We shall construct a non-deterministic turing machine which checks if the number of (D,R) pairs satisfying assessment is greater than a given number.

Let the input of the non-deterministic turing machine to be constructed as follows.

#g# all the (D,R) pairs, their assessment and a partial solution

where g is a number.

The non-deterministic turing machine guesses which terms are in which classes. Then it checks if the number of (D,R) pairs satisfying assessment is greater than g. If it is, then it accepts; otherwise, it rejects. This non-deterministic turing machine runs in polynomial time because by definition of non-determinism the guess is always correct and the time required to check the number of (D,R) pairs satisfying assessment is polynomial.

By hypothesis, there is an equivalent deterministic turing machine running in deterministic polynomial time.

Let us give this deterministic turing machine the following input.

#k-i# all the (D,R) pairs, their assessment and a partial solution

where k is the number of (D,R) pairs, and i=1.

The deterministic turing machine must halt in polynomial time and either accepts or rejects the input. If it accepts, then by the construction of the turing machine, all the (D,R) pairs are satisfied. If it does not accept, then i is incremented to 2. The above process is repeated until the least value of i is found such that the deterministic turing machine accepts the input. Note that when i = k + 1, the turing machine must accept and thus the process takes no more than polynomial time. When the least value of i is found such that the turing machine accepts, the maximum number of (D,R) pairs satisfying assessment = (k - i + 1).

From the above theorems, we obtain the following results.

Theorem 3.1 The satisfiable assessment problem with a partial solution is polynomial complete.

References

[1] G. Salton: Computer evaluation of indexing and text processing. J. Ass. Comput. Mach., 1968, 15, 8.

[2] K. Sparck Jones and D.M. Jackson: Current approaches to classification and clump-finding at the CLRU. Comput. J., 1967, 10, 29.

[3] L.B. Doyle: Is automatic classification a reasonable application of statistical analysis of text? J. Ass. Comput. Mach., 1965, 12, 473.

[4] D.M. Jackson: The construction of retrieval environments and pseudo-classifications based on external relevance. Inform. Stor. Retr. 1970, 6, 187.

[5] S.A. Cook: The complexity of theorem proving procedures. Conf. Record of Third ACM Symposium on Theory of Computing. 1971, 151.

[6] S.K. Sahni: On the knapsack and other computationally related problems. Ph.D. thesis, Cornell University, August 1973.

[7] G. Salton: The Smart Retrieval System - Experiments in Automatic Document Processing. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1971.